Data Streams

Sucking up message packets

Jan 2, 1989

Introduction

There are instances of the need to access data streams in the VME system. This note explores a generic method of data request for such data streams.

Serial input data

One data stream that has already been supported is the serial input data stream. A request is made for serial data using a specific listype with an ident which gives the serial port that is being accessed. Typically, such a request is made at 15 Hz, specifying a maximum size of the response buffer. The response data to the request is a data structure consisting of a word containing the number of bytes read from the serial port followed by the bytes themselves. If the number of bytes = 0, then no serial data was collected since the last time. When the number of bytes > 0, the data bytes comprise an integral number of lines of data with each terminated by a carriage return, limited by the size of the requested data. Both nulls and linefeeds are ignored in serial input and do not appear in the buffer.

In the current implementation of serial data requests, only one requester can access a given serial port at once; reading the serial input queue is "destructive." It would be better if more than one requester could sample the serial input at the same time. A data stream protocol can allow for this.

Ramp readback data

Another type of data stream in the VME systems is a ramping co-processor's readback data. In this case, a circular buffer is continuously filled by the co-processor with readings taken at 720 Hz, for example. Requests for this data must be supported. In this case, it may be sufficient to access only the data that is placed into the queue *after* the request is made.

Diagnostic data

One might imagine a diagnostic queue whose contents are to be sampled by a host level diagnostic program. In this case, we need to be able to access data written into the queue *before* the request is made.

Data stream handling

If there were a system table that housed pointers to such data stream queues, then a listype could be designed which addressed one of these streams and sampled what data it found there, returning it at the requested rate.

Response data format

For a general purpose data stream, let us assume that packets are written into the queue consisting of a size word followed by (size-2) bytes of data. This is exactly the same format used for command packets sent to a co-processor command queue. The response data returned in the requester's buffer would consist of an integral number of such packets. If the size word were zero, no more packets would follow in the response buffer. If there were no data found in the queue, then the first word would be zero, which is entirely consistent with the general format.

Queue types

In order to deal with more than one format of data stream, one can either use different listypes or standardize on a header format which includes a type value. In this way, the processing can vary depending on the queue type. Examples would use different header

formats, use pointers rather than offsets, and use both forward and backward "links" to support looking at previous entries in the queue.

Options for data stream access

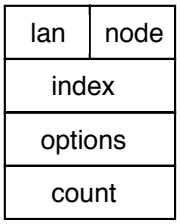
There are several possibilities for copying out the data from the queue to the requester. One case might be to read a raw bytes stream. This could be suitable for a low level access to serial port data, for example. There would be no editing of the byte stream, nor would there be any automatic assembly into lines of text. All such higher level logic would be handled by the requester.

A second example might be access to the raw data, stripping the size word(s) placed in the buffer. The size would have to be assumed known and constant for this to make sense.

And one might like to read old data—that which was placed into the queue before the request was made. To do this, we need to specify how much old data is requested. It could be specified as a number of packets or as a number of bytes. For packets, we need to be using a queue format that includes two "size" words per packet. The first would be the size of the packet, as usual. The second would be the relative offset from the start of the packet (the first size word) to the start of the previous packet. In this way, one can work backwards packet by packet to see previous entries, allowing for the packets to be of different sizes. If the packets were of equal size, and if the offset to the packet located latest in memory were recorded in the queue header, we could look backwards without these size words. But there must be a parameter in the request which specifies either the number of bytes of prior data or the number of packets of prior data.

Ident format

Consider the following ident format:

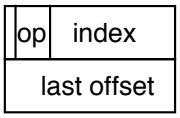


The first word is the lan-node as usual. The second word is the index which selects the desired data stream. The third word selects the options for data collection from the queue. And the fourth word is the count of packets or bytes of prior data requested.

Request processing

In order to allow sampling of the data stream which is non-destructive, one must keep the "last" pointer within the request data block. Currently, there is a single 4-byte pointer used to fulfill a data request associated with each listype-ident pair. To handle data streams, this pointer must keep track of the location of the last entry. But how can it know how to circle back to the start of the buffer? It would seem that 32 bits is not enough.

As a solution, suppose the pointer value which is used contains two word-size values. The first is an index into the data stream pointer table. This was given by the ident and would normally be converted into a pointer to the entry in the table. By refraining from converting it to a pointer we save two bytes, which can be used as the second word to contain the offset to the last data packet read from the queue. Now the internal pointer value can lead us to the queue (or data stream) header as well as the position in the data stream where we last left off.



To keep the options value, let the entire second word of the ident be used as the first word of the internal pointer. Then the option value is kept internally to the request. But we can only keep 3 bits for this purpose, as the most significant bit of the first word of the pointer must be used to mark a pointer to an external answer fragment buffer contained within the request.

Data Stream Pointer Table

A system table is used to house pointers to data stream queues. This table is indexed by the index value referred to above.